An Efficient Index for Reachability Queries in Public Transport Networks

Tesfaye, Bezaye ; Augsten, Nikolaus ; Pawlik, Mateusz ; Böhlen, Michael Hanspeter ; Jensen, Christian S

Abstract: Computing path queries such as the shortest path in public transport networks is challenging because the path costs between nodes change over time. A reachability query from a node at a given start time on such a network retrieves all points of interest (POIs) that are reachable within a given cost budget. Reachability queries are essential building blocks in many applications, for example, group recommendations, ranking spatial queries, or geomarketing. We propose an efficient solution for reachability queries in public transport networks. Currently, there are two options to solve reachability queries. (1) Execute a modified version of Dijkstra's algorithm that supports time-dependent edge traversal costs; this solution is slow since it must expand edge by edge and does not use an index. (2) Issue a separate path query for each single POI, i.e., a single reachability query requires answering many path queries. None of these solutions scales to large networks with many POIs. We propose a novel and lightweight reachability index. The key idea is to partition the network into cells. Then, in contrast to other approaches, we expand the network cell by cell. Empirical evaluations on synthetic and real-world networks confirm the efficiency and the effectiveness of our index-based reachability query solution.

DOI: https://doi.org/10.1007/978-3-030-54832-2_5

An Efficient Index for Reachability Queries in Public Transport Networks

Bezaye Tesfaye[1], Nikolaus Augsten[1], Mateusz Pawlik[1], Michael H. Böhlen[2], and Christian S. Jensen[3]

[1] University of Salzburg, Austria
{bezaye.belayneh, nikolaus.augsten, mateusz.pawlik}@sbg.ac.at

[2] University of Zurich, Switzerland
boehlen@ifi.uzh.ch

[3] Aalborg University, Denmark
csj@cs.aau.dk

Abstract. Computing path queries in public transport networks is challenging because the shortest path between nodes changes over time according to schedule. A reachability query from a node at a given start time on such a network retrieves all points of interests (POIs) that are reachable within a given cost budget. Reachability queries are essential building blocks in many applications, for example, group recommendations, ranking spatial queries, or geomarketing. In this paper, we propose an efficient solution for reachability queries in public transport networks. Currently, there are two options to solve reachability queries. (1) Execute a modified Dijkstra's algorithm that supports time-dependent edge traversal costs; this solution is slow since it must expand edge by edge and does not use an index. (2) Issue a separate path query for each single POI, i.e., a single reachability query requires answering many path queries. None of these solutions scales to large networks with many POIs. We propose a novel and lightweight reachability index. The key idea is to partition the network into cells. Then, in contrast to other approaches, we expand the network cell by cell. Empirical evaluations on synthetic and real world networks confirm the efficiency and the effectiveness of our index.

Keywords: Reachability Queries, Public Transport Networks, Temporal Graphs, Spatial Network Databases

1 Introduction

In this paper we deal with the problem of reachability queries in public transport networks. A reachability query retrieves all points of interest (POIs) reachable from a given query node for specific start time and time budget. The start time is required since the reachability result changes over time.

As an application senario, consider a platform that recommends events to a group such that the people like the event and attend it together [2,13]. When

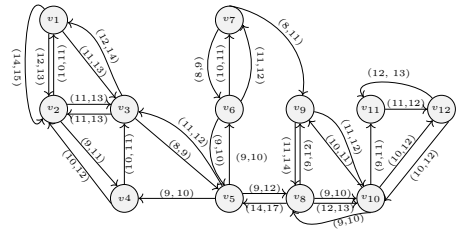
the group is given, the events must be evaluated by various criteria to optimize the benefit for the given group. One important aspect is the location of the event relative to the group members. Events that are too far away for some of the group members are unlikely to be successful. In another scenario, the event is given and a group, that would enjoy attending this event together, must be formed. Here, in addition to the benefit of the event for a group, also the compatibility of the people to each other must be evaluated. When forming the group, reachability is again an important issue. In the most general scenario, both the group and the event are to be recommended. In each of these scenarios, multiple reachability queries are required to generate a single recommendation.

Another example is a real estate website that ranks properties according to user preferences. The user may specify reachability criteria, for example, for schools, the working place, or other POIs. Thereby, the time budget for individual types of POIs may vary: a user may be willing to commute to work for an hour, but schools should be nearby. Further, houses for real estate applications or hotels in a booking portal may be ranked by reachable infrastructure like grocery stores, pharmacies, hospitals, restaurants, or shopping areas. Ranking the results of a single user query requires the computation of multiple reachability queries. Similarly, in urban planing or when opening new franchise stores, multiple reachability queries are required to optimize the location of a new facility.

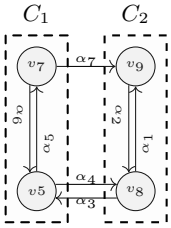
To support these applications, reachability queries must be computed efficiently. Achieving this goal in public transport networks is tricky since the shortest path between two nodes depends on the starting time, and the time to traverse a path may greatly vary over time. In a public transport network, stations are nodes and connections between stations are edges between nodes. An edge can only be traversed at specific points in time given by the schedule. Therefore, computing an index for public transport networks is more complex than for networks with constant edge-traversal cost or networks in which an edge can be traversed at any time (like pedestrian or road networks).

Example 1. *Consider, the public transport network in Figure 1a. The nodes* $v_1, v_2, \ldots, v_{12}$ *represent stations and the directed edges represent connections between the stations. Each connection has a pair* (t_d, t_a) *of departure and arrival times. For example, there is a connection leaving* v_4 *at time 10 and arriving at* v_3 *at 11. The cost of traversing the edge* (v_4, v_3) *at time 9 is 2, since we have a waiting time in addition to the edge traversal time. The shortest path from* v_{10} *to* v_{11} *at start time* $t_s = 9$ *has cost 2 (edge* (v_{10}, v_{11})*), while at* $t_s = 10$ *the cost of the shortest path is* 3 *(edges* (v_{10}, v_{12})*,* (v_{12}, v_{11})*). At start time* $t_s = 9$ *the nodes* $\{v_8, v_9, v_{11}\}$ *are reachable from* v_{10} *with budget* $\Delta t = 2$*; at* $t_s = 10$ *with the same budget we can reach the nodes* $\{v_9, v_{12}\}$*.*

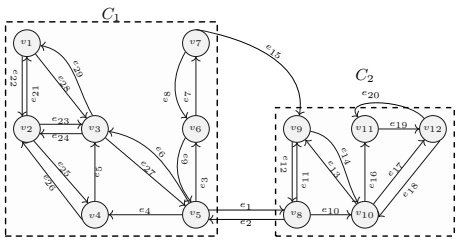
The state of the art in answering reachability queries in public transport networks includes two approaches. The first approach is a temporal Dijkstra's algorithm [10] that expands in the network until the budget is used up. Such algorithms compute a so-called isochrone (the reachable area) and intersect it with the set of POIs [6,11]. Since all edges in the isochrone must be expanded,



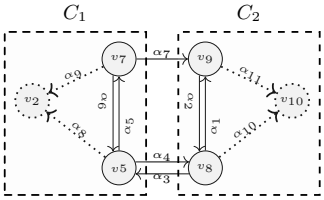
(a) A public transport network.



(c) Core of the reachability index R on temporal graph G.



(b) Temporal graph G of the network in (a). G is partitioned into cells C_1, C_2.



(d) Reachability index R after inserting two POIs, v_2 and v_{10} (dotted nodes and edges).

Fig. 1: Temporal graph of public transport network and reachability index.

these algorithms do not scale to large networks. The second approach translates a single reachability query into a set of path queries (e.g., shortest path or earliest arrival path [16,18,19]), one for each POI. These approaches require heavy index structures and do not scale to a large number of POIs.

In this paper, we propose an index-based technique for reachability queries in public transport networks. Instead of expanding edge by edge, in a precomputation step, we partition the network into cells and construct a novel reachability index. At query time, the index is used to expand cell by cell. Each cell covers a region of the network and all POIs in that region. The precomputation effort for a specific cell is independent of the other cells such that the index scales to large networks and is often smaller than the original graph.

To the best of our knowledge, this is the first work that proposes an index for reachability queries in public transport networks.

The rest of the paper is structured as follows. In Section 2 we define the problem and give an overview of our solution in Section 3. We introduce our reachability index in Section 4 and discuss query processing using the index in Section 5. In Section 6 we review the related works. In Section 7 we showcase the effectiveness of our index in an experimental setting. We conclude in Section 8.

2 Preliminaries and Problem Definition

In a public transport network, stations are nodes and connections are edges. A connection has a departure time t_d and an arrival time t_a. We assume periodic schedules as is typically the case in public transport networks, e.g., schedules repeat daily or weekly.

A *temporal graph* $G = (V, E, c)$ is a directed graph with vertices V, edges $E \subseteq V \times V$, and a time-dependent cost function $c(e, t)$, $c : E \times \mathbb{R} \to \mathbb{R}_{\geq 0}$, for traversing an edge e starting at time t. We represent public transport networks as temporal graphs with a specific cost function, which we derive from the schedule. Each station is a node in the graph, and there is an edge from node u to node v iff there is a direct connection (i.e., there are no intermediate stops) from the station of u to the station of v. The cost function is periodic with period Π, i.e., $c(e, t) = c(e, t + \Pi)$ and piecewise linear; all linear pieces have slope $k = -1$; the cost function is not continuous; all discontinuities are at departure times of some connections. For a single connection $s_i = (t_d, t_a)$ on an edge e, the cost in the period $(t_d - \Pi, t_d]$ is $c_i(e, t) = t_a - t$; if there are multiple connections $S = \{s_1, s_2, \ldots, s_i\}$ for edge e, the cost of e at time t is the minimum of all costs of the individual connections at time t, $c(e, t) = \min\{c_i(e, t) \mid c_i \text{ is the cost function of connection } s_i\}$. Our cost function is *consistent*, i.e., for any edge $e \in E$ and all start times $t_1 \leq t_2$: $t_1 + c(e, t_1) \leq t_2 + c(e, t_2)$. Intuitively, in a consistent cost function it never pays off to wait. Consistency is required for the use with Dijkstra's shortest path algorithm [15].

Example 2. *Consider the edge e_7 in Figure 1b with connections $s_1 = (10, 11)$ and $s_2 = (11, 12)$. Then, for $\Pi = 12$, the cost function of s_1 is $c_1(e_7, t) = 11 - t$, $t \in (-2, 10]$ and for s_2 is $c_2(e_7, t) = 12 - t$, $t \in (-1, 11]$. The overall cost function $c(e_7, t)$ is the minimum of $c_1(e_7, t)$ and $c_2(e_7, t)$.*

A *path* p from u to v in a temporal graph $G = (V, E, c)$ is a sequence of edges, $p = \langle e_1, e_2, \ldots, e_n \rangle$, such that $e_i \in E$, $e_i = (w_{i-1}, w_i)$, $w_0 = u$, and $w_n = v$; $P(u, v)$ is the *set of all paths* from node u to node v. The cost of a path is the fastest time to traverse the path at a given the start time. Due to the consistency property of our cost function, the path cost is the sum of all edge costs. The *cost of path* $p = \langle e_1, e_2, \ldots, e_n \rangle$ at time t, is the cost sum of all edges in p: $c(p, t) = \sum_{1 \leq i \leq n} c(e_i, t_i)$, where $t_1 = t$ and $t_i = t_{i-1} + c(e_{i-1}, t_{i-1})$ for $i > 1$. The *shortest path cost* from node u to node v at time t is the minimum cost of a path from u to v, $sp(u, v, t) = \min\{c(p, t) \mid p \in P(u, v)\}$. A path with the minimum cost is called the *shortest path*. A node v is *reachable* from a node u at time t within budget Δt iff there is a path $p \in P(u, v)$ such that the cost of p at time t is not larger than Δt, i.e., $c(p, t) \leq \Delta t$. The *reachability query*, $RQ(u, t, \Delta t) = \{v \in V \mid \exists p \in P(u, v), c(p, t) \leq \Delta t, v \in POI\}$, in a temporal graph $G = (V, E, c)$ with points of interst $POI \subseteq V$, returns all points of interest reachable from node u at time t within budget Δt.

Problem definition The goal of this work is to develop an efficient index-based solution for reachability queries that scales to large temporal graphs.

3 Solution Overview

We propose a novel index structure, the *reachability index*, to answer reachability queries. We introduce a bulk loading technique for our index, provide access methods for answering reachability queries, and discuss incremental insertions and deletions of POIs in the index.

The reachability index for the temporal graph is build in a precomputation step. To construct the index, we partition the graph into cells. The index is a temporal graph that contains only those nodes of the original graph that are POIs or directly connect different cells, called *border nodes*. Each POI belongs to a cell. We keep edges between border nodes of neighboring cells and introduce new edges between the border nodes within a cell. Further, an edge between each POI and the border nodes in its cell is introduced. The edge costs are the costs of shortest paths between the respective nodes in the original graph. POIs can be dynamically inserted to and deleted from the index at any time.

A search query traverses the index cell by cell. The border nodes are used to cross cells and to reach neighboring cells. For each border node, we verify if any of the POIs in that cell is reachable.

4 The Reachability Index

The reachability index R is a temporal graph that is constructed from the original graph G. Compared to the original graph, the index has fewer nodes and allows for faster expansion. The index construction proceeds as follows:

1. *Graph partitioning.* The nodes of graph G are split into disjoint *cells*. At query time, instead of expanding edge by edge in G, we expand cell by cell in the index.
2. *Node and edge insertion.* Based on the partitioning into cells, we insert new nodes and new edges into the index.
3. *Cost function.* The edge cost is computed as a shortest path cost for each departure time from a source node to a destination node.
4. *Inserting POIs.* Inserting a POI into a cell adds a new node and edges to the border nodes of the cell. POIs that are not *border nodes* can be inserted and deleted dynamically without modifying the rest of the index.

The choice of "good" cells renders the index more effective, but does not affect its correctness. We define requirements for a good partitioning and propose a suitable partitioning technique.

4.1 Graph Partitioning

We partition the nodes of a temporal graph $G = (V, E, c)$ into a set of disjoint cells $C = \{C_1, C_2, \ldots, C_n\}$, such that each node of G belongs to exactly one cell C_i, i.e., $C_i \cap C_j = \emptyset$ for any pair of cells with $i \neq j$, and $\bigcup_{1 \leq i \leq n} C_i = V$.

Each disconnected component of the graph should be partitioned into at least two cells. Within each cell C_i, we distinguish *border nodes* B_i. A node $v \in C_i$ is a border node if it has an edge to or from another cell, i.e., there is a node $w \in V, w \notin C_i$, and an edge $(v, w) \in E$ or an edge $(w, v) \in E$.

The cells define the structure of our reachability index. The index will be expanded cell by cell to answer reachability queries. A good partitioning should satisfy the following properties:

1. *Well connected inside.* The cells should comprise highly-linked nodes with many edges and connections.
2. *Loosely connected outside.* The number of border nodes per cell should be small).
3. *Large distance between cells.* The number of connections between neighboring partitions should be small at query time.

Finding a good partitioning that satisfies our requirements is not straightforward. In our scenario, the number of partitions or their sizes is not known upfront, which rules out many partitioning techniques. We propose to use Louvain method for community detection [7], which produces good partitions in our tests. This technique efficiently finds communities in a network. It partitions the graph into communities of strongly connected nodes; nodes from different communities are loosely connected. The quality of the partitions is the so-called modularity which measures the density of links inside a community as compared to links between communities. Louvain iteratively finds good communities by increasing the modularity value. It starts with each node being in a different community and improves by moving nodes between communities. It provides a weight function for the links between nodes. We chose the number of connections between nodes, i.e., how many times, according to the schedule, one can cross a direct edge between two nodes.

Example 3. *Figure 1b shows the temporal graph G of the public transport network in Figure 1a and its partitioning into two cells: C_1 and C_2 (the dashed boxes). v_5 and v_7 are border node of C_1. v_8 and v_9 are border nodes of C_2.*

4.2 Constructing the Index Core

Given a temporal graph $G = (V, E, c)$ and a partitioning C of G, we construct the core of our reachability index. The index core is independent of POIs and never changes. The reachability index is a temporal graph $R = (V_R, E_R, c_R)$ with nodes $V_R \subset V$, edges $E_R \subseteq V_R \times V_R$, and cost function $c_R(e, t)$ on the edges $e \in E_R$. For an edge $e = (u, v) \in E_R$, c_R returns the shortest path cost from u to v at time t, i.e., $c_R(e, t) = sp(u, v, t)$.

Index nodes For each cell $C_i \in C$, we insert all its *border nodes* B_i into the node set V_R of the index. Thus, the nodes of the index $V_R = \bigcup_{1 \leq i \leq |C|} B_i$.

Index edges The edges of the index core are $E_R = BB \cup BC$, such that:

- BB are all edges between border nodes of neighboring cells. For each edge $(u, v) \in E$ between two border nodes of different cells in C, $u \in C_i, v \in C_j, i \neq j$, insert a new edge between the respective nodes into the index, $E_R = E_R \cup \{(u, v)\}$.
- BC are edges between pairs of border nodes within a cell. For each pair $u, v \in B_i$, insert two new edges (u, v) and (v, u) into the index, $E_R = E_R \cup \{(u, v), (v, u)\}$.

Example 4. *Figure 1c shows the index core of the temporal graph in Figure 1b. It contains all border nodes in* C_1, $\{v_5, v_7\}$, *and in* C_2, $\{v_8, v_9\}$; $BB = \{\alpha_3, \alpha_4, \alpha_7\}$, $BC = \{\alpha_1, \alpha_2, \alpha_5, \alpha_6\}$.

4.3 Edge Cost

The cost function c_R of an edge $e = (u, v) \in E_R$ in index R is defined as the shortest path cost from u to v at time t in graph G, i.e., $c_R(e, t) = sp(u, v, t)$.

Computing the index cost function For computing the values of the cost function c_R, we execute Dijkstra's single source shortest path algorithm once for every border node $b \in B_i$ and every departure time at b. The expansion stops when all other border nodes in the cell and all direct neighbors of b (i.e., nodes reachable from b via a BB edge) are visited. Since the cells are small compared to the overall graph, typically only a small number of nodes needs to be considered for each execution of Dijkstra.

Example 5. *In the index core shown in Figure 1c, example costs for edges in* BB *are* $c(\alpha_3, 9) = 3, c(\alpha_4, 8) = 3, c(\alpha_7, 8) = 2$, *and costs for edges in* BC *are* $c(\alpha_1, 9) = 2, c(\alpha_2, 11) = 3, c(\alpha_5, 9) = 2, c(\alpha_6, 8) = 2$.

4.4 Points of Interest

POIs can be inserted and deleted at any time, also after index construction. This is beneficial because POIs may change over time. A POI $v \in V$ may be any node in the original temporal graph. If v is a border node, no action is required because such node is in the index core already. Otherwise, similarly to border nodes, inserting v to the index has three steps. (1) We add v to the index nodes ($V_R = V_R \cup \{v\}$). (2) We add an edge from each border node of $v's$ cell to v (we call such edges BP edges). (3) The cost function based on shortest paths (like for all other edges) is computed. Deleting a POI from the index removes the POI node and all its incoming edges.

Example 6. *Figure 1d shows a reachability index of the temporal graph* G *in Figure 1b after inserting two POIs* v_2, v_{10} *with edges* $BP = \{\alpha_8, \alpha_9, \alpha_{10}, \alpha_{11}\}$. *Example costs of edges in* BP *are* $c(\alpha_8, 9) = 3$, $c(\alpha_9, 8) = 7$, $c(\alpha_{10}, 9) = 1$, $c(\alpha_{10}, 12) = 1$, *and* $c(\alpha_{11}, 11) = 1$.

4.5 Index size

The index consists of border nodes and POIs. Thus, the number of index nodes is at most the number of nodes in the temporal graph. We introduce three types of edges into the index. BB edges connect border nodes between different cells and they are the subset of the temporal graph edges. BC edges connect border nodes in a single cell and their number is at most quadratic in the number of border nodes. Each POI add as many BP edges as border nodes in a cell. The number of both, BC and BP edges, depends only on a subset of temporal graph nodes that are in a single cell. It does not depend on the graph size. Each edge has as many edge cost values as departure times from a node. The edge costs are computed for a single cell and independently from other cells. Hence, their computation can be highly parallelized. In particular, the edge costs from a single border node at one departure time is computed independently from other costs.

4.6 Index Compaction

The index size, as well as the size of the temporal graph, is dominated by the size of the schedule, i.e., the number of edge connections. After computing the edge costs in the index, we observe that many different departure times have the same arrival time in the destination. It is enough to keep only one connection per arrival time, the one with maximum departure time. We leverage that and compact the index by reducing the number of connections as follows. Consider an edge $e(u, v) \in E_R$ and set S of departure–arrival connection pairs (d, a) on that edge. We reduce S to $S' \subseteq S$, such that $S' = \{(d, a) \in S : \nexists_{(d_i,a_i)\in S} a_i = a \wedge d_i < d\}$. As we show in our experiments this compaction technique is highly effective and reduces the index size up to 73% (cf. Section 7).

5 Answering Reachability Queries

The core idea of our reachability algorithm is to expand cell by cell rather than expanding edge by edge. In addition, we discuss two aspects of reachability queries: a heuristic to avoid unnecessary edge expansions and the processing of query nodes that are non-border nodes.

The reachability algorithm takes the index R, query node q, start time t_s, the cost budget Δt, and the cost function c as the input. The expansion proceeds similar to Dijkstra's shortest path algorithm and returns a set of reachable nodes in index R. To retrieve the correct edge cost we do a binary search in the list of edge costs which we sort by departure time. We only need to consider nodes and edges from the reachability index, i.e., border nodes, POIs, and all edges between them. The BB edges between border nodes of different cells allow us to expand to the neighboring cells; the BC edges between border nodes of the same cell reflect the time to cross a cell; the direct BP edges from border nodes to POIs allow for a quick evaluation of which POIs can be reached. We terminate the algorithm when no more nodes within the remaining cost budget can be found.

Example 7. *Consider reachability index in Figure 1d and the edge costs from Examples 5 and 6. A reachability query from v_5 at start time $t = 8$ with cost budget 6 retrieves POIs v_2 and v_{10}, where $sp(v_5, v_2, t) = 4$ (through α_8) and $sp(v_5, v_{10}, t) = 5$ (through α_4 and α_{10}). For start time $t = 6$, the only reachable POI is v_2, because $sp(v_5, v_2, t) = 6$ (through α_8) but $sp(v_5, v_{10}, t) = 7$ (through α_4 and α_{10}).*

Avoiding unnecessary expansions Regarding the edges within a cell we observe the following. Consider the Dijkstra's algorithm processing a border node b of a cell C_i. Then, the cost of the other nodes within the cell, $v_j \in C_i$, are updated w.r.t. the cost of reaching them from b. When we pop a node v_j in a later round, and if v_j was last updated by b, there is no point in following the edges from v_j to the other nodes in the cell. The cost of accessing the other nodes in the cell through v_j cannot be smaller than the cost of accessing these nodes directly from b since all edge costs are shortest paths. If, however, v_j was updated through a an edge from a neighboring cell, the edges to the other nodes in the cell need to be followed.

We leverage this observation and avoid following edges inside a cell that cannot lead to an update and thus do not affect the solution. We flag the nodes whenever their cost was updated by processing a node from within a cell, and remove the flag otherwise. The outgoing edges that must be expanded are selected based on the flag.

Note that the number of edges within a cell is quadratic in the number of border nodes of that cell. Thanks to the use of the flag we avoid unnecessary expansions. In particular, if the cheapest way to reach all nodes in a cell is through k border nodes, we only expand $k(w-1)$ edges per cell, where w is the number of all border nodes and POIs in a cell. The value of k is expected to be small and will often be $k = 1$ (i.e., the shortest path from a query node q to all nodes in the cell crosses the border node that is closest to q).

Non-border query nodes The reachability index does not contain all nodes of the original graph. If the query node q in cell C_i is not a border node, we calculate the costs $c((q, b'), t_s)$ to all border nodes in the cell, $b' \in B_i$. Then, we continue the expansion in the index from each border node b' at time $t_s + c((q, b'), t_s)$.

Correctness We show that the shortest path costs in the index and the original temporal graph are identical. Let $u, w \in V_R$ be two index nodes and $p = \langle (v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n) \rangle$ the corresponding shortest path in the temporal graph, $u = v_0, w = v_n$. If there is a direct edge between u and w in the index, the shortest path is the cost of that edge: this cost is precomputed using Dijkstra for each departure time in the original temporal graph; since our cost function is consistent (cf. Section 2), the edge cost is correct [15]. Otherwise, u and w are not in the same cell (all nodes in a cell are connected with an edge). So, there must be a path along index nodes $u_1, u_2, \ldots u_k \subseteq v_1, \ldots v_{n-1}$ that are all on path p since cells can be left only through border nodes. We show that the cost of the index path is indeed the shortest path. Assume a node u_i such

that $sp(v_0, v_n, t) < sp(u, u_i, t) + sp(u_i, w, t + sp(u, u_i, t))$. On a path of length two, costs of edges (u, u_1) and (u_1, w) are precomputed shortest path costs, and therefore correct. The assumption, however, implies that one of the edge costs could be decreased, i.e., is incorrect. This argument can be extended edge by edge to paths of arbitrary length.

6 Related Work

Shortest path and reachability queries on road networks, i.e., graphs with constant edge cost, have been extensively studied. Unfortunately, these works cannot be readily applied to public transport networks [3]. The evaluation of Bast [4] shows the big performance gap between the two types of networks. This is due to the time-dependent edge cost of public transportation networks, which makes the precomputation effort of many algorithms infeasible.

Current solutions for public transport networks either rely on Dijkstra's algorithm [10] or require heavy precomputations. Examples for Dijkstra-based approaches are isochrone algorithms for multimodal networks [6,11]. They expand from a query point using Dijkstra and compute a so-called isochrone, which is the reachable portion of the network at a given point in time. Since all edges in the isochrone must be expanded, this approach does not scale to large networks.

Many works fall into the category of labelling approaches. The earliest work, 2-hop labels [9], is designed for weighted graphs and is based on 2-hop covers of shortest paths. Recent works strive to decrease the index size and construction time [8,14], which form the bottleneck of 2-hop labels and prevent their application to large graphs. Time Table Labelling (TTL) [18] and Top Chain [19] adapt 2-hop labels to public transport networks; they support shortest path and point-to-point reachability queries. In TTL, the main idea is to precompute label sets for each node v containing reachable nodes from and to v. Top Chain creates a directed acyclic graph (DAG), where each node represents a departure time, and decomposes the DAG to create the label sets. Creating label sets in both techniques requires high precomputation costs and large index sizes. To decrease the index size, Top Chain only stores K label sets, called chains. The index size of Top Chain for small K values is smaller than that of TTL, but there is no guarantee that the query results can be found using the index.

Non-labelling techniques include Scalable Transfer Patterns [5], Connection Scan Algorithm (CSA) [17], and Contraction Hierarchy for Timetables (CHT) [12]. Transfer Patterns require an expensive profile search from each node to find the optimal paths to all other nodes. CSA organizes a schedule as two sequences of edges. The first sequence contains sorted edges based on arrival times, the second sequence sorts edges based on departure times. All these approaches depend on expensive precomputations and/or large index sizes that limit their scalability.

To compute reachability queries as defined in this paper, all techniques based on point-to-point queries require the computation of shortest paths from a given query node to every POI, which does not scale to a large number of POIs.

Table 1: Statistics of our datasets

Dataset	#Nodes	#Edges	#Conn	#Part	#B-nodes		Part. size			#POIs	
					sum	avg	avg	min	max	sum	avg
Zurich	2508	5630	555713	45	315	7.0	55	2	157	99	2.20
Berlin	12984	34791	1348070	50	1241	24.8	259	2	921	567	11.34
Synthetic	145188	433272	31042468	44	1245	28.3	3299	831	4037	7176	163.00

7 Experiments

We experimentally evaluate our solution, *RQ*, and compare it with two competitors, a no–index solution, *NI*, and a fully indexed solution, *SP*. We report on the index size and efficiency of the algorithms w.r.t. the number of expanded edges, which is the work that an algorithm has to do to find reachable nodes.

Competitors The no–index solution, *NI*, operates on the original temporal graph and does not build an index. The reachability is computed with a modified version of Dijkstra's algorithm that supports our cost function (cf. Section 5). The fully–indexed solution, *SP*, stores all shortest paths from every node in the temporal graph to all POIs at every departure time. SP represents the collection of works that index the shortest paths between pairs of nodes (cf. Section 6).

Implementation The algorithms are implemented in Python 3 and executed on a Intel Xeon E5-2630 v3 2.40GHz server with 2 CPUs, 8 cores each (hyperthreading enabled), 96GB of RAM, running Debian 9.12.

Datasets We use two real-world public transport networks represented as temporal graphs, *Zurich* and *Berlin* [1], and one *Synthetic* graph. *Zurich* and *Berlin* are obtained as open data in GTFS format that we further processed. For these graphs, we chose all transport modes and all connections operating on Mondays. *Synthetic* is a 6×6 grid of equally–sized spider-web subgraphs. Each spider-web subgraph has one edge to every neighboring subgraph (to its left, right, top, and bottom). With this graph we simulate loosely connected cities that are densely connected inside.

The statistics of the data graphs are shown in Table 1. #Conn is the number of all connections (departure–arrival pairs) that can be used to cross an edge. Additionally, we show the statistics of partitioning the data graphs using the Louvain method (with maximum partition sizes): number of partitions, number of border nodes (sum and average per partition), partition sizes (avg, min, and max). We also show the number of POIs (sum and average per partition). POIs are chosen randomly as 5% of the nodes of each partition (at least one per partition).

7.1 Index size

RQ and *SP* precompute certain shortest paths and build an index structure that is sufficient to answer reachability queries. If the index of *SP* is stored as

Table 2: Index details

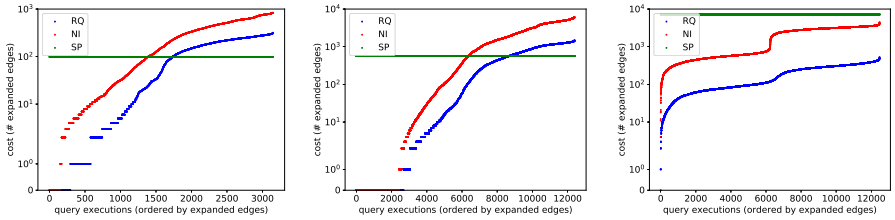
Dataset	Algorithm	#Nodes	#Edges	#Connections
Zurich	*RQ*	414	4021	421268
	SP	2508	248292	55015587
	NI	2508	5630	555713
Berlin	*RQ*	1808	53543	2533940
	SP	12984	7361928	764355690
	NI	12984	34791	1348070
Synthetic	*RQ*	8421	212564	18018811
	SP	145188	1041869088	222760750368
	NI	145188	433272	31042468

a graph, its number of nodes equals #Nodes (POIs are nodes of the graph), its number of edges equals #Nodes × #POIs (shortest paths from every node to every POI are computed), and the number of connection equals #Conn × #POIs (a shortest path at every departure time to every POI is computed); #Nodes, #POIs, and #Conn being of the original temporal graph. Although *NI* does not require precomputation, the input graph has to be kept in memory. In Table 2, we compare the index sizes (*RQ*, *SP*) to the input graph size (*NI*). The values that increase the index size are the number of nodes and edges, and the number of connections at the edges. The index size of *RQ* is smaller in every aspect than that of *SP* (up to four orders of magnitude), for the original *Zurich* and *Synthetic* graphs (*NI*); for the *Berlin* dataset the index of *RQ* is not significantly larger than the original input graph. #Connections is the number of edge connections stored. For *RQ*, we list the absolute number of connections resulting after the compaction. The reduction rate of compaction varies from 67% in *Synthetic* to 73% in *Zurich* and *Berlin*.

7.2 Number of expanded edges

To evaluate the efficiency of our solution, in Figure 2 we compare the number of edges that an algorithm has to process in order to find all reachable POIs. One data point in the figure (scatter plot) is a single reachability query. Data points are sorted along the x–axis by the number of expanded edges. The number of expanded edges (y–axis) is displayed in log scale. We execute one reachability query starting at every border node in our index, at five different start times (8:00, 12:00, 16:00, 18:00, 22:00) for two time budgets (60 and 120 minutes). Thus, the number of data points is 10×#Border nodes. The budget is chosen large enough to force *RQ* to traverse multiple edges. Since the edge costs of large cells in the *RQ* index are often above 15 minutes (and above 30 minutes in about half of the cases), budgets near these values provide little insight.

Since *SP* precomputes the path to each POI, it always evaluates one edge per POI. This is a lower bound on the cost of any point-to-point index. Although the index of *SP* is orders of magnitude larger, *RQ* expands significantly fewer edges for many of the data points. We observe the largest differences for the budget



(a) Zurich (b) Berlin (c) Synthetic

Fig. 2: Number of expanded edges (y-axis in log scale).

of 120 minutes. On the *Synthetic* dataset, the number of expanded edges by RQ is up to three orders of magnitude lower than that of SP, and up to an order of magnitude lower than NI. RQ always expands fewer edges than NI. Values equal to zero indicate that an algorithm cannot expand due to high connection costs. We also performed similar experiment with an increased percentage of POIs (more than 5%): the difference in the number of expanded edges between RQ and NI decreases. This is to be expected since RQ can leverage the sparsity of POIs, while NI cannot.

Overall, our experiments show that - dispite its small size - RQ substantially reduces the number of edges (by about an order of magnitude in realistic settings) and therfore speeds up reachability queries in public transport networks.

8 Conclusion

In this paper, we dealt with reachability queries in temporal graphs that retrieve all points of interest (POIs) that are reachable from a user-defined query node, start time, and time budget. We observe that current solutions do not scale when either the network grows large (solutions without precomputed index that are based on Dijkstra's shortest path algorithm) or there are many POIs in the network (solutions based on an index for single-path queries which must be executed for each POI separately). We proposed a solution based on a novel access structure, the reachability index. Our index partitions the original temporal graph into cells. This enabled us to expand the graph cell by cell rather than edge by edge. Our experiments suggest that our technique is both effective and efficient.

References

1. Zurich and Berlin GTFS. https://data.stadt-zuerich.ch/dataset/vbz_fahrplandaten_gtfs, https://daten.berlin.de/datensaetze/vbb-fahrplandaten-gtfs (Accessed January 31, 2020)
2. Amer-Yahia, S., Roy, S.B., Chawlat, A., Das, G., Yu, C.: Group recommendation: Semantics and efficiency. Proc. VLDB Endow. (2009). https://doi.org/10.14778/1687627.1687713

3. Bast, H.: Car or public transport - two worlds. In: Efficient Algorithms, Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday. vol. 5760, pp. 355–367 (2009). https://doi.org/10.1007/978-3-642-03456-5_24
4. Bast, H., Delling, D., Goldberg, A.V., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., Werneck, R.F.: Route planning in transportation networks. In: Algorithm Engineering - Selected Results and Surveys (2016). https://doi.org/10.1007/978-3-319-49487-6_2
5. Bast, H., Hertel, M., Storandt, S.: Scalable transfer patterns. In: Workshop on Algorithm Engineering and Experiments, ALENEX (2016). https://doi.org/10.1137/1.9781611974317.2
6. Bauer, V., Gamper, J., Loperfido, R., Profanter, S., Putzer, S., Timko, I.: Computing isochrones in multi-modal, schedule-based transport networks. In: ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems (2008). https://doi.org/10.1145/1463434.1463524
7. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment (10) (2008). https://doi.org/10.1088/1742-5468/2008/10/p10008
8. Cheng, J., Huang, S., Wu, H., Fu, A.W.: Tf-label: a topological-folding labeling scheme for reachability querying in a large graph. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD (2013). https://doi.org/10.1145/2463676.2465286
9. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. SIAM J. Comput. (2003). https://doi.org/10.1137/S0097539702403098
10. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. (1959). https://doi.org/10.1007/BF01386390
11. Gamper, J., Böhlen, M., Cometti, W., Innerebner, M.: Defining isochrones in multimodal spatial networks. In: ACM Int. Conf. on Information and Knowledge Management. CIKM (2011). https://doi.org/10.1145/2063576.2063972
12. Geisberger, R.: Contraction of timetable networks with realistic transfers. In: Experimental Algorithms, 9th Int. Symposium, SEA. (2010). https://doi.org/10.1007/978-3-642-13193-6_7
13. Jameson, A., Smyth, B.: Recommendation to groups. In: The Adaptive Web, Methods and Strategies of Web Personalization (2007). https://doi.org/10.1007/978-3-540-72079-9_20
14. Jin, R., Wang, G.: Simple, fast, and scalable reachability oracle. PVLDB **6** (2013). https://doi.org/10.14778/2556549.2556578
15. Kaufmann, D.E., Smith, R.L.: Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. Journal of Intelligent Transportation Systems (1993). https://doi.org/10.1080/10248079308903779
16. Seufert, S., Anand, A., Bedathur, S.J., Weikum, G.: FERRARI: flexible and efficient reachability range assignment for graph indexing. In: IEEE Int. Conf. on Data Engineering, ICDE (2013). https://doi.org/10.1109/ICDE.2013.6544893
17. Strasser, B.: Intriguingly simple and efficient time-dependent routing in road networks. CoRR (2016)
18. Wang, S., Lin, W., Yang, Y., Xiao, X., Zhou, S.: Efficient route planning on public transportation networks: A labelling approach. In: ACM SIGMOD Int. Conf. on Management of Data, SIGMOD (2015). https://doi.org/10.1145/2723372.2749456
19. Wu, H., Huang, Y., Cheng, J., Li, J., Ke, Y.: Reachability and time-based path queries in temporal graphs. In: IEEE Int. Conf. on Data Engineering, ICDE (2016). https://doi.org/10.1109/ICDE.2016.7498236